Design and Implementation of a Quadruple Floating-point Fused Multiply-Add Unit

He Jun
Shanghai Hi-Performance IC Design Centre
Shanghai, China
e-mail: joyhejun@126.com

Zhu Ying
Shanghai Hi-Performance IC Design Centre
Shanghai, China
e-mail: zhuying_1116@sina.com

***Abstract*—High precision and high performance floating-point unit is an important research object of high performance microprocessor design. According to the characteristic of quadruple precision (QP) floating-point data format and research on double precision floating-point fused multiply-add (FMA) algorithms, a high performance QPFMA is designed and realized, which supports multiple floating-point arithmetic with a 7 cycles pipeline. By adopting dual adder and improving on algorithm architecture, optimizing leading zero anticipation and normalization shifter logic, the latency and hardware cost is decreased. Based on 65nm technology, the synthesis results show that the QPFMA can work at 1.2GHz, with the latency decreased by 3 cycles, the gate number reduced by 18.77% and the frequency increased about 11.63% comparing to current QPFMA design, satisfying the requirements of high performance processor design.**

Keywords-floating-point arithmetic; fused multiply-add; quadruple precision; high precision

I. INTRODUCTION

IEEE-754 floating-point double precision (64-bit) or extended double precision (80-bit) arithmetic has been implemented in modern processors, but which is not sufficient for many scientific applications, such as climate modeling, supernova simulations, coulomb N-body atomic system simulations, electromagnetic scattering theory, computational geometry and grid generation, computational number theory and so on[1]. As floating-point data presentation and arithmetic are not precise, the error is increased gradually after many times of floating-point arithmetic, resulting that the computation results are imprecise and incredible.

In order to improve the precision and reoccurrence of floating-point results and enhance the stability of numerical algorithms, higher precision floating-point computation is required. The IEEE 754-2008 floating-point standard [1]has included quadruple precision (QP) floating-point data type (binary128) to support high precision floating-point computation. And it also takes the floating-point fused multiply-add (FMA) operation as one of the basic operations. So it has become one of the hot topics on how to design and implement high precision and high performance floating-point unit efficiently.

Base on the research on traditional double precision floating-point FMA arithmetic algorithms, considering of the characteristic of the new QP floating-point data type, a new high performance QPFMA unit is designed and realized, which supports floating-point multiplication, addition and comparison operation besides four types of floating-point FMA operations [1] (multiply-add, multiply-sub, negative multiply-add and negative multiply-sub). The QPFMA mainly comprises of a 113-bit Booth2 multiplier, a 341-bit alignment shifter, a 342-bit dual adder, a 228-bit leading zero anticipation (LZA) block, a 342-bit normalization shifter and a rounding processing block, with 7-stage pipeline and 7-cycle operation latency.

After optimization on the algorithm and detailed analysis on the critical timing path, the register transistor level (RTL) design of the QPFMA is synthesized using 65nm cell library, which can work at 1.2GHz. Compared to the design of paper [3], operation latency of the QPFMA is decreased by 3 cycles with the frequency increased by 11.63% and the gate number is also reduced by 18.77%. As the results show that the QPFMA can satisfy the requirements of high performance processor design.

This paper proceeds as follow. Firstly, section II introduces QP floating-point data format and related work on QP floating-point arithmetic. Secondly, the design of QPFMA is presented in section III, including the overall architecture and key components. Thirdly, the implementation of QPFMA is described in section IV. Finally, the conclusion is provided.

II. RELATED WORK

A. QP Floating-Point Data Format

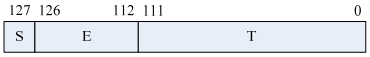

Figure 1. QP floating-point data format

As the IEEE 32-bit single precision (SP) and 64-bit double precision (DP) data type, IEEE 754-2008 floating-point standard [1]defines 128-bit QP floating-point data type (binary128), consisting of three fields as the 1-bit sign (S), the 15-bit exponent (E) and 112-bit fraction (T) (see Figure 1.). The bias of exponent is 16383. The fraction of normalized data implies the integer bit as 1, which is not need to presented, so the precision is 113-bit in fact (see TABLE I.).

[1] multiply-add: F=AB+C; multiply-sub: F=AB-C; negative multiply-add: F=-AB+C; negative multiply-sub: F=-AB-C.

TABLE I. IEEE 754-2008 FLOATING-POINT DATA TYPE

Floating-point Data Type	SP	DP	QP
Sign (S) width	1	1	1
Exponent (E) width	8	11	15
Fraction (T)width	23	52	112
Precision (p) width	24	53	113
Data width	32	64	128
bias	127	1023	16383

B. *Related Research on QP*

According to the requirements of scientific computation for high precision and high reliable floating-point arithmetic, there are many researches on QP floating-point arithmetic in academic field. The algorithms of QP floating-point multiplication [4](with a 3-cycle pipeline), add [5](with a 3-cycle pipeline) and division [6](with the operation latency of 59 cycles, no pipelined) have been studied, which all supporting two operation modes: one QP or two parallel DP floating-point operations. A multifunction floating-point FMA with a 4-cycle pipeline was provided in paper [7], which supports one QP/DP or two parallel DP/SP floating-point FMA and one dot product operation. A similar design of floating-point FMA was designed in paper [8], supporting one QP or two parallel DP operations with a 3-cycle pipeline. To decrease hardware overheads, the multiplier block in paper [8]only supports DP multiplication and produces QP multiplication by iterating one more time, so the throughput of QP operation is half. A pure 5-cycle pipelined QPFMA was implemented in paper [9], which can work at 202MHz synthesized in SMIC 0.13um technology. Later, the design is optimized by extending to 10 pipeline stages in paper [10], which can work at 465MHz synthesized in SMIC 0.13um technology and 1.075GHz synthesized in TSMC 65nm technology. Besides, a QPFMA was implemented using FPGA in paper [10], which can accelerate LU and MGS-QR decomposing by 42 to 97 times while producing higher precision results and dissipating lower energy.

As to our known, mainstream microprocessors support QP floating-point arithmetic with limited hardware, mainly with software emulation. Only IBM POWER6 processor support QP floating-point addition in hardware [11]. Although QP floating-point data type and operations are define in SPARC v9 instruction set, there is no SPARC processor supports QP floating-point in hardware as to now. There are some libraries on QP and higher precision floating-point arithmetic, such as Intel Fortran [12], GMP, MP-FR[13], QD (Quad-Double) [14]and so on. But the performance of software emulation is limited. Compared to DP floating-point, the test time of QP floating-point LINPACK is increase by 35 times [1].

In brief, supporting QP floating-point arithmetic is one of the important trends of floating-point unit, helping to increase the precision of floating-point arithmetic and improve the performance of some important scientific applications. Current academic study on QP floating-point arithmetic put emphasis on theory exploration, far from industry implementation. Considering of hardware and latency, the implementation QP floating-point arithmetic is not attractive for big area and long latency. So the key issue is to decrease the hardware overhead and operation latency of QPFMA.

III. HIGH PERFORMANCE QPFMA DESIGN

A. *Overall Architecture*

Floating-point FMA algorithm is realized in IBM RS/6000 [15] for the first time, which has been looked as the classic algorithm of FMA, made up of following steps (mainly the fraction part):

1) Fractions multiplication: Multiplicand and multiplier is multiplied, generating carry-save form's product (Carry, Sum).

2) Addend alignment: Paralleled with multiplication, according to the exponent difference of addend and product, the addend is shifted by an alignment shifter to make the exponents equal.

3) Sum of product and addend: Firstly add the aligned addend and product using a 3:2 carry save adder (CSA) to get a new carry-save form's product (Carry`, Sum`), and then add them to get the sum using a carry propagated adder (CPA), probably needing to complement the sum if negative to get a positive result.

4) Result normalization: While sum of product and sum, the leading one is predicted by a LZA, and then the result is shifted by a normalization shifter to get normalized result.

5) Result rounding: Based on the IEEE rounding mode, the normalized result is rounded and the last result is obtained, may needing another renormalization if the last result greater than or equal to 2.0.

On the basis of classic FMA algorithm, there are many studies to reduce the latency of FMA operation, which can be classified into two types: one is the algorithm optimized by normalization in advance and combined addition of CPA and result rounding [16], the other is dual-path or multi-path algorithm [17][18][19]. Such improved algorithms can reduce the FMA latency, but at the cost of increasing hardware area and complex.

To reduce the hardware cost and latency, the QPFMA algorithm in this article is optimized on the basis of classic FMA algorithm, including the application of dual adder to avoiding the complement of negative result and improved LZA logic with decreased data width and levels of normalization shifter.

The QPFMA unit is mainly comprised of a 113-bit multiplier, a 341-bit alignment shifter, a 342-bit dual adder, a 228-bit LZA, a 342-bit normalization shifter and a rounding block with 7 pipeline stages (as shown in Figure 2. , the dash-dotted lines ST0~6 mean the flip-flops of the pipeline stages).

The function of every stage of the QPFMA is described as following:

ST0: Operands are unpacked and selected based on operation types at first, while determining the effective operation is subtraction or addition and whether need to invert the last result's sign. And the special operands, such as zero, not-a-number, de-normalized number and infinity, are all detected and the input exception result is generated. According to the exponent of each operand (E_A、E_B、E_C),

the exponent difference (d)[2]and the alignment shifter count (ASC)[3] are calculated out, and then make sure if d greater than one or not and get the temporary result exponent (E_{tmp}). The multiplier finishes Booth recoding and produces 57 part products, and then the part products is compressed by first level 4:2 CSAs.

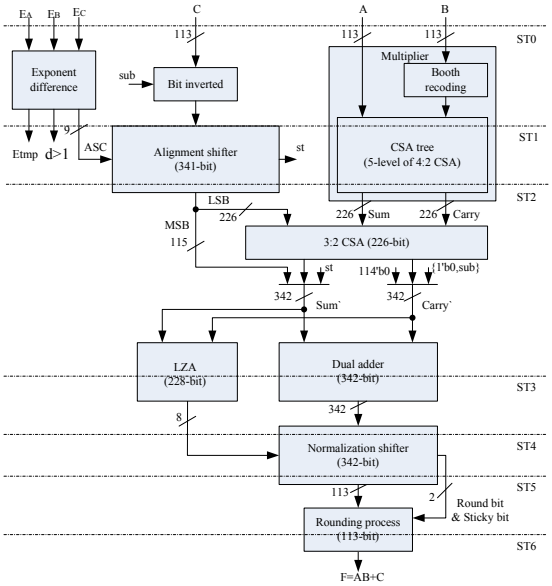

Figure 2. QPFMA Overall Architecture

ST1: The addend is shifted in line with ASC and the sticky bit for rounding is also produced. The multiplier continues to compress the part products using the 2nd, 3rd and 4th level 4:2 CSAs.

ST2: The multiplier accomplishes the last level part product compression and produce the carry-save product (Carry, Sum). Then the carry-save product and aligned addend is compressed by a 3:2 CSA, generating the new carry-save sum (Carry`, Sum`). Finally, the carry-save sum is added by a dual adder while predicting the leading one position of the last result by LZA (only finish partly).

ST3: The rest of dual adder and LZA accomplished paralleled, and the preliminary result is obtained with the normalization shifter count. Then the preliminary result is shifted for the first level.

ST4: The remaining part of normalization shifter is finished and the normalized result is got while adjusting the exponent accordingly.

ST5: As there is a few errors in LZA, the normalized result need to be corrected and then the result is rounded based on IEEE rounding mode.

ST6: The rounded result is detected to determine if overflow or underflow occurs or not. And the last FMA result is selected from the rounded result and former generated input exception result in ST0.

2 $d=E_C-E_{AB}=E_C-bias-(E_A-bias+E_B-bias)=E_C-E_A-E_B+bias$.

3 $ASC=E_{AB}-(E_C-116)=116-(E_C-E_{AB})=116-d$.

The input exception processing block is not shown in Figure 2, which detecting input operand exceptions and generating exception result, for example, invalid operation and infinity operation result.

In the following the key components is described in detailed, including the 113-bit Booth2 Multiplier, alignment shifter, dual adder, LZA and normalization shifter.

B. Key Components

1) 113-bit Booth2 Multiplier

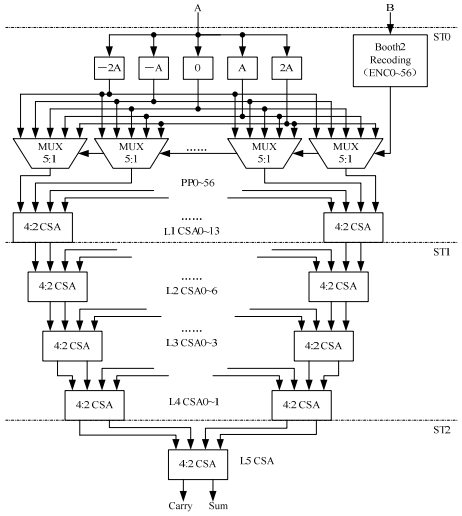

Figure 3. 113-bit Booth2 Multiplier

The 113-bit multiplier is made up of two parts: Booth2 recoding and part products compression tree and divided into 3 pipeline stages ST0~3 (see Figure 3.). After Booth recoding, 57 part products are generated firstly, and then compressed into 29 part products by 14 4:2 CSAs of the first level. Secondly the 29 part products are compressed into 15 part products by 7 4:2 CSAs of the second level and then into 8 part products by 3 4:2 CSAs and a 3:2 CSA of the third level. Thirdly the 8 part products are compressed into 4 part products by 2 4:2 CSAs of the forth level. Finally the 4 part products are compressed into 2 part products (Carry, Sum) by the last 4:2 CSA.

2) Alignment Shifter

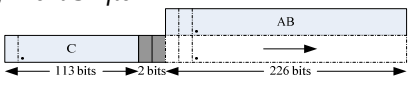

Figure 4. QPFMA Alignment

For the product of AB is acquired only after the multiplication finished while the addend C is ready at first, the alignment shifter only shift the addend C while AB is fixed. Assuming that the addend C is shifted left for 116 bits, then the addend C only need to shifted right according to ASC(see Figure 3.). If ASC≤0 C need not to be shifted, else it is shifted ASC bits right. But when ASC>341, meaning that AB is far greater than C and C is only need to taken into account to form sticky bit (st), it is shifted right for maximum 341 bits. So ASC is in the range of [0, 341].

The sticky bit (st) is generated as following: If the effective operation is addition, the st bit is formed by "OR"ing all the bits of C shifted out beyond 341. While if the effective operation is subtraction, the st bit is formed by "AND"ing all the bits of C shifted out beyond 341 after C is bit inverted. After the st bit is generated, it is jointed to the least significant bit (LSB) of the Sum` output from the 3:2 CSA while a bit of "sub" (sub=1 if effective operation is subtraction) jointed to the LSB of the Carry` output from the 3:2 CSA. Then both the correct sticky bit and complement of the addend C can be realized when Sum` and Carry` are added.

The alignment shifter consists of 4 levels of 4:1 multiplex and 1 level of 2:1 multiplex with the data width of 341 bits and alignment count in the rage of [0, 341]. It is not detailed more for conciseness.

3) Dual Adder

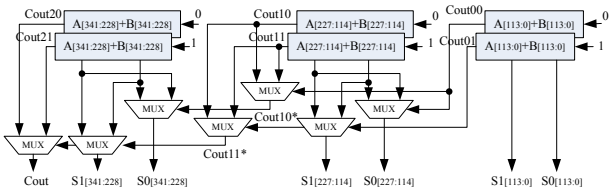

Figure 5. 342-bit Dual Adder

The input A and B of the dual adder are the output (Carry`, Sum`) of 3:2CSA, which are added to produce the sum (S0 and S1) and carry out (Cout). Considering the long latency of 342-bit adder, the dual adder is designed incorporated both the idea of end-around-carry adder and carry select adder. It is divided into 3 segments, including two parallel 114-bit adders with the same addends and carry in as 0 and 1 in each segment (see Figure 5.).

First, the six 114-bit adders calculate in parallel and get each carry out (Cout). Then according to the carry out (Cout00 and Cout01) of the low segment, the sum (S0 and S1) and carry out (Cout10* and Cout11*) of the middle segment is determined. And then the sum (S0 and S1) and carry out (Cout) of the high segment is determined according to the carry out Cout10* and Cout11*). Finally, S0 is bit inverted to get the final result if the effective operation is subtraction or S1 is the final result if not.

The dual adder can calculate out the absolute of (Carry`, Sum`) directly, avoiding the complement of negative result and reducing the latency of a 342-bit adder. The latency of the dual adder is the latency of a 114-bit adder and a 2:1 multiplex.

4) LZA and Normalization Shifter

After analysis on the FMA algorithm, it is concluded that the leading one's position in fraction must be in the least significant 228-bit of the result and the most significant 114-bit must be all zeroes if exponent difference no larger than one (i.e. d≤1), while the leading one's position in fraction must be in the most significant 228-bit of the result if d>1. So LZA is carried out on the least significant 228-bit if d≤1 while on the most significant 228-bit if not, resulting the data width of LZA decrease from 342 bits down to 228 bits.

Accordingly, there are 3 cases for the temporary result exponent (E_{tmp}). 1) When d≤1 (i.e. ASC≥115), E_{AB}+2 is taken as the temporary result exponent (E_{tmp}). 2) When d>1 and ASC≤0, E_{tmp} equals to E_C. 3) In the rest case (i.e. d>1 and 0<ASC<115), E_{tmp} equals to E_C+ASC.

Normalization shifter normalizes the fraction by shifting left according to the result of LZA, making the MSB of result fraction be one. On the base of former analysis, the most significant 114 bits must be all zeroes if d≤1, so the result is left shifted 114 bits first and then is left shifted according to the result of LZA. If d>1, the leading one of fraction must be in the most significant 228-bit of the result, and the result is only need to be left shifted according to the result of LZA directly. Finally, the shifted result probably needs to be left shifted one more bit for the error of LZA.

All the processing described above fits for the case of effective subtraction. While in the case of effective addition, it is processed as following when the result is normalized:

1) If d≤1, $E_{tmp}=E_{AB}+2$, the result is left shifted 114 bits for the most significant 114 bits must be all zeroes, and then the result is not shifted any more. In this case, the most significant 2 bits of shifted result maybe zero, meaning that error of 2 bits exist, which need to be corrected later.

2) If 1<d<116 (i.e. 0<ASC<115), $E_{tmp}=E_{AB}$+ASC, the result is left shifted ASC-1 bits. There is error of 1-bit in this case, needing to be corrected later.

3) If d≥116 (i.e. ASC≤0), $E_{tmp}=E_C$, the result is need not to be left shifted and no error exists.

So in the case of effective addition, the result is left shifted only when 0<ASC<115, and there may exist error of 2-bit needing to be corrected later.

To sum up, the normalization shifter divides into two parts: one part is a 114-bit shifter first and the other is a normal left shifter, shifting result according to the result of LZA result in case of effective subtraction while shifting result according to ASC in case of effective addition. And there exist error of 2-bit at most, which can be corrected by detecting the most significant 2-bit of the normalized result to determine whether they are zero or not. Using the LZA and normalization shifter designed here, the data width of LZA logic reduces 114 bits and the normalization shifter avoids one level of multiplex, helping to reduce hardware overhead and operation latency.

IV. HIGH PERFORMANCE QPFMA IMPLEMENTATION

A. Logic Synthesis and Estimation

Taking use of the 65nm cell library, the RTL Verilog code of the QPFMA is synthesized into gate netlist using the design compiler EDA tool of Synopsys. As the synthesis result shows, the QPFMA is balanced pipelined with the cycle time of 0.74ns and the critical timing path lies in the second pipeline stage. Considering the overhead of timing from the clock port (CK) to the output port (Q) and the skew of clock, the QPFMA can work at 1.2GHz frequency, satisfying the demand of high performance microprocessor.

The QPFMA designed in paper [3]has a 10-stage pipeline with the frequency at 1.075GHz in TSMC 65nm technology. Compared to it, the QPFMA in this paper reduces the latency

by 3 cycles, decrease the gate number by 18.77% and increase the frequency by 11.63% or so (see TABLE II.).

TABLE II. QPFMA SYNTHESIS RESULTS

Design	Pipeline stages	Frequency (GHz)	Gate number	Area (μm2)
QPFMA in paper[3]	10	1.075	229000	unknown
QPFMA in this paper	7	1.2	186007	334811.41
DPFMA in paper[3]	8	1.086	93830	unknown
DPFMA in this paper	6	1.2	83084	149550.60

Moreover, the QPFMA only increase one pipeline stage when compared with the former designed double precision FMA (DPFMA) by us. The area of QPFMA is 2.24 times of the area of the DPFMA when both work at 1.2GHz. As described in paper [3], the area of the QPFMA designed in it is 2.44 times of the area of the DPFMA in the same technology and at similar frequency.

V. CONCLUSION

In conclusion, a high performance QPFMA unit is designed and synthesized in this paper, and it is compared with other QPFMA. The main contribution of this paper is as following: 1) A new QPFMA with 7-stage pipeline is put forward, which can work at 1.2GHz in 65nm technology, satisfying the requirement of high performance processor; 2) By the application of dual adder, avoiding the complement of negative result, and the optimization LZA logic, reducing the data width of LZA and the level of normalization shifter, the hardware overhead and operation latency of the QPFMA is decrease both. The QPFMA will be optimized further by improvement in algorithm to decrease area and latency more.

REFERENCE

[1] Bailey D H. High-precision floating-point arithmetic in scientific computation. Computing in Science and Engineering, 2005, 7(3):54-61.

[2] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. IEEE Standard 754-2008, 3 Park Avenue New York, NY 10016-5997, USA 29 August 2008.

[3] Li Tie-jun, Li Qiu-liang, Xu Wei-xia, A High Performance Pipeline Architecture of 128bit Floating-point Fused Multiply-add Unit, JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY 2010, 32(2):56-60.

[4] A. Akkas and M.J. Schulte. Dual-Mode Floating-Point Multiplier Architectures with Parallel Operations. Journal of Systems Architecture, 52:549-562, 2006.

[5] A. Akkas. Dual-Mode Quadruple Precision Floating Point Adder. In 9th Euromicro Conference on Digital System Design, pages 211-220, 2006.

[6] A. Akkas. A Dual-Mode Quadruple Precision Floating-Point Divider. Fortieth Asilomar Conference on Signals, Systems and Computers, pages1697-1701,2006.

[7] Mustafa Gok, Metin Mete Ozbilen. Multi-functional floating-point MAF designs with dot product support, Microelectronics Journal, 2008, 39(1):30-43.

[8] Libo Huang, Sheng Ma, Li Shen, Zhiying Wang, Nong Xiao, Low-Cost Binary128 Floating-Point FMA Unit Design with SIMD Support, IEEE TRANSACTIONS ON COMPUTERS, VOL.61, NO.5, pp745-751, MAY 2012.

[9] Zhang Feng, Li Tie-jun, Xu Wei-xia, Research and Realization of a 128-Bit Multiply-Add-Fused Unit, COMPUTER ENGINEERING AND SCIENCE, 2009, 31(2):93-103.

[10] Lei Yuan-wu, Dou Yong, Guo Song, High Precision Scientific Computation Accumulator on FPGA, Chinese Journal of Computers, 2012, 35(1):112-122.

[11] Xiao Yan Yu; Yiu-Hing Chan; Curran, B.; Schwarz, E.; Kelly, M.; Fleischer, B. A 5GHz+ 128-bit Binary Floating-Point Adder for the POWER6 Processor, Proceedings of the 32nd European Solid-State Circuits Conference, pages 166-169, 2006.

[12] Intel Company, Intel Compilers and Libraries, http://soft-ware.intel.com/en-us/articles/intel-cimpilers/. 2012/12/24.

[13] Fousse L, Hanrot G, Lefevre V et al. Mpfr: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software (TOMS), 2007,33(2):1-14.

[14] Hida Y, Li X S, Bailey D H. Quad-double arithmetic: Algorithms, implementation, and application. Lawrence Berkeley National Laboratory, Berkeley, CA, Report LBL-46996,2000.

[15] R. K. Montoye, et. al., Design of the IBM RISC System/6000 Floating-Point Execution Unit, IBM Journal of Research and Development, Vol. 34, pp. 61-62, 1990.

[16] T. Lang and J. Bruguera, Floating-Point Fused Multiply-Add with Reduced Latency, Internal Report, Dept. of Electronic and Computer Engineering University of Santiago de Conpostela (Spain), Jan. 2002(available at www.ac.usc.es).

[17] Bruguera, J.D.; Lang, T. Floating-point fused mulipy-add: reduced latency for floating-point addition, Proc. 17th IEEE Symp. Computer Arithmetic, Hyannis, 27-29 June, 2005.

[18] Seidel, P.M. Multiple path IEEE floating-point fused multiply-add, Proc. 46th Int. IEEE Midwest Symp. Circuits and Systems (MWS-CAS), 2003.

[19] Eric Quinnell, Floating-Point Fused Multiply-Add Architectures. Phd thesis,University of Texas at Austin, May 2007.